(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平6－242924

(43)公開日　平成６年(1994)９月２日

(51)Int.Cl.[5]	識別記号	庁内整理番号	Ｆ Ｉ	技術表示箇所
Ｇ０６Ｆ　5/00		Ｈ　9189－5B		
12/12		Ｚ　7608－5B		
Ｈ０３Ｍ　7/30		Ｚ　8522－5Ｊ		

審査請求　未請求　請求項の数８　ＦＤ　（全 14 頁）

(21)出願番号　特願平5－283976

(22)出願日　平成５年(1993)10月18日

(31)優先権主張番号　９６３，２０１

(32)優先日　1992年10月19日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　590000400
ヒューレット・パッカード・カンパニー
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000

(72)発明者　チャールス・ジェイ・ローゼンバーグ
アメリカ合衆国カリフォルニア州パロ・アルト・グリーア・ロード3201

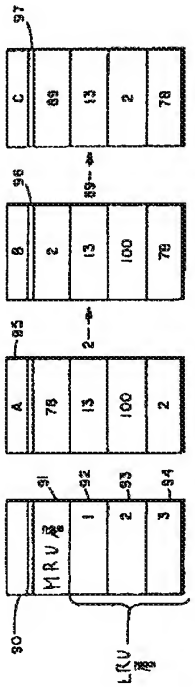
(72)発明者　トーマス・ジー・バーグ
アメリカ合衆国アイダホ州ボイジィ・イースト・ジェファーソン・ストリート702

(54)【発明の名称】　データ圧縮／伸長システム及び方法

(57)【要約】　(修正有)

【目的】相異なる出現パターンの個数が辞書の容量を超えた場合に、捨てるべきパターンを決めるに当たって、完全なＬＲＵ管理を行うと辞書管理のための計算量の負担が大きい点を改善する。

【構成】出現パターンの辞書を作り、与えられたパターンが辞書中のパターンと一致した場合には、そのパターンの代わりに辞書中のパターンの位置を示す識別子を出力することによってデータ圧縮を行う。辞書を最も最近使用されたパターンが入る第１のキャッシュ部分91と、それ以外の第２のキャッシュ部分92-94に分ける。新たなパターンが与えられると、それを第１の部分へ入れ、以前第１の部分に入っていたパターンを第２の部分中でランダムに選択された部分に上書きする。



【特許請求の範囲】
【請求項１】以下の(a)ないし(d)を設けたデータ圧縮システム：
(a)複数のレベルを有する第１のキャッシュ・メモリ手段：第１の前記レベルは最も最近使用されたデータ・セグメントに割り当てられ、第２の前記レベルは最も最近使用されたものではないデータ・セグメントに割り当てられる；
(b)受信したデータ・セグメントが前記第１のキャッシュ・メモリ手段中にあるか否かを判定する手段：前記受信したデータ・セグメントが前記第１のキャッシュ・メモリ手段中になかった場合には、前記受信したデータ・セグメントを前記第１のレベルに割り当てられていた以前に受信していたデータ・セグメントに代えてそこに割り当て、前記以前に受信していたデータ・セグメントを最も最近使用されたものではないデータ・セグメントに代えて前記第２のレベルの中の１つの位置に割り当て、前記代えられる最も最近使用されたものではないデータセグメントは前記第２のレベルに割り当てられている最も最近使用されたものではないデータ・セグメントの中から疑似ランダム方法によって選択される；
(c)前記第１のキャッシュ・メモリ手段中にはなかった旨の表示及び前記受信したデータ・セグメントを圧縮されない形式で受信手段に送出する手段；
(d)受信したデータ・セグメントが前記第１のキャッシュ・メモリ手段中にある旨の判定に応答して、前記送出手段をして、前記受信手段に対して、前記第１のキャッシュ・メモリ手段中のどこに前記受信したデータ・セグメントが割り当てられているかを指示する位置コードを送出させる手段。
【請求項２】以下の(a)ないし(e)を設けたデータ圧縮／伸長システム：
(a)複数のレベルを有する第１のキャッシュ・メモリ手段：第１の前記レベルは最も最近使用されたデータ・セグメントに割り当てられ、第２の前記レベルは最も最近使用されたものではないデータ・セグメントに割り当てられる；
(b)受信したデータ・セグメントが前記第１のキャッシュ・メモリ手段中にあるか否かを判定する手段：前記受信したデータ・セグメントが前記第１のキャッシュ・メモリ手段中になかった場合には、前記受信したデータ・セグメントを前記第１のレベルに割り当てられていた以前に受信していたデータ・セグメントに代えてそこに割り当て、前記以前に受信していたデータ・セグメントを最も最近使用されたものではないデータ・セグメントに代えて前記第２のレベルの中の１つの位置に割り当て、前記代えられる最も最近使用されたものではないデータセグメントは前記第２のレベルに割り当てられている最も最近使用されたものではないデータ・セグメントの中から疑似ランダム方法によって選択される；
(c)前記第１のキャッシュ・メモリ手段中にはなかった旨の表示及び前記受信したデータ・セグメントを圧縮されない形式で受信手段に送出する手段；
(d)受信したデータ・セグメントが前記第１のキャッシュ・メモリ手段中にある旨の判定に応答して、前記送出手段をして、前記受信手段に対して、前記第１のキャッシュ・メモリ手段中のどこに前記受信したデータ・セグメントが割り当てられているかを指示する位置コードを送出させる手段；
(e)前記受信手段は以下の(e-1)及び(e-2)を含み、圧縮された形態のデータを伸長する：
(e-1)複数のレベルを有する第２のキャッシュ・メモリ手段：前記レベルのうちの第１のレベルは最も最近使用されたデータ・セグメントに割り当てられ、前記レベルのうちの第２のレベルは前記最も最近使用されたデータ・セグメントよりも長い間使用されていない複数のデータ・セグメントに割り当てられている；
(e-2)前記第２のキャッシュ・メモリ手段に関連するデータ・セグメントを維持・管理してそこに割り当てられたデータが前記第１のキャッシュ・メモリ手段中のデータ・セグメントと同期し提示・管理されるように前記擬似ランダム法を行う手段。
【請求項３】以下のステップ(a)ないし(f)を設け、ラスタ・スキャン表現の２レベルのデータを複数のキャッシュ記憶を使用して圧縮し、各キャッシュ記憶は前記ラスタ・スキャン表現からの複数のデータ・セグメントを使用状況の階層に基づいてストアするデータ圧縮方法：
(a)キャッシュ・コンテキスト識別子を前記複数のキャッシュ記憶の各々に割り当てる：前記キャッシュコンテキスト識別子の各々は前記ラスタ・スキャン表現の１つのスキャン行中のデータ・セグメントに関連する値を有する；
(b)前記１つのスキャン行の次のスキャン行に含まれている現データ・セグメントにアクセスする：前記現データ・セグメントの各々は前記１つのスキャン行中のコンテキスト・データ・セグメントに位置的に整列している；
(c)前記現データ・セグメントの各々を位置的に整列しているコンテキスト・データ・セグメントと同じ値が割り当てられたキャッシュ記憶中に含まれるデータ・セグメントと比較する；
(d)前記現データ・セグメントがキャッシュ記憶中にあった場合には当該キャッシュ記憶中の識別子位置を送出し、もし前記現データ・セグメントが前記キャッシュ記憶中になかった場合には見つからなかった旨の識別子及び前記現データ・セグメントを送出する；
(e)現データセグメントが前記次のスキャン行中に含まれ、次回のデータ・セグメントのスキャン行についてのコンテキスト・データ・セグメントになるようにする；
(f)他のどのキャッシュ記憶にも新たなコンテキスト・

セグメント値が割り当てられていない場合には前記新たなコンテキスト・セグメント値を新たなキャッシュ記憶に割り当て、処理すべきスキャン行がそれ以上なくなるまでステップ(b)ないしステップ(f)を繰り返す。

【請求項４】以下のステップ(a)ないし(h)を設け、ラスタ・スキャン表現の２レベルのデータを複数のキャッシュ記憶を使用して圧縮し、各キャッシュ記憶は前記ラスタ・スキャン表現からの複数のデータ・セグメントを使用状況の階層に基づいてストアし、更に前記圧縮されたデータを伸長するデータ圧縮／伸長方法：

(a)キャッシュ・コンテキスト識別子を前記複数のキャッシュ記憶の各々に割り当てる：前記キャッシュコンテキスト識別子の各々は前記ラスタ・スキャン表現の１つのスキャン行中のデータ・セグメントに関連する値を有する；

(b)前記１つのスキャン行の次のスキャン行に含まれている現データ・セグメントにアクセスする：前記現データ・セグメントの各々は前記１つのスキャン行中のコンテキスト・データ・セグメントに位置的に整列している；

(c)前記現データ・セグメントの各々を位置的に整列しているコンテキスト・データ・セグメントと同じ値が割り当てられたキャッシュ記憶中に含まれるデータ・セグメントと比較する；

(d)前記現データ・セグメントがキャッシュ記憶中にあった場合には当該キャッシュ記憶中の識別子位置を送出し、もし前記現データ・セグメントが前記キャッシュ記憶中になかった場合には見つからなかった旨の識別子及び前記現データ・セグメントを送出する；

(e)現データセグメントが前記次のスキャン行中に含まれ、次回のデータ・セグメントのスキャン行についてのコンテキスト・データ・セグメントになるようにする；

(f)他のどのキャッシュ記憶にも新たなコンテキスト・セグメント値が割り当てられていない場合には前記新たなコンテキスト・セグメント値を新たなキャッシュ記憶に割り当て、処理すべきスキャン行がそれ以上なくなるまでステップ(b)ないしステップ(f)を繰り返す；

(g)複数のキャッシュ記憶を確立する：前記確立されたキャッシュ記憶の各々は前記ラスタ・スキャン表現の第１のスキャン行中のデータ・セグメント値に関連する値を持つ割り当てられたコンテキスト識別子を有する；

(h)伸長された選考スキャン行中のコンテキスト識別子に基づいてキャッシュ記憶を選択し、前記選択されたキャッシュ記憶中の前記位置識別子によって指示される位置内容から、前記位置に割り当てられたデータ・セグメントが何であるかを判定することによって、受信した位置識別子を伸長する。

【請求項５】以下のステップ(a)ないし(f)を設け、データ・セグメントのストリームを複数のキャッシュ記憶を使用して圧縮し、各キャッシュ記憶は前記データ・ストリームからの複数のデータ・セグメントを使用状況の階層に基づいてストアするデータ圧縮方法：

(a)キャッシュ・コンテキスト識別子を前記複数のキャッシュ記憶の各々に割り当てる：前記キャッシュコンテキスト識別子の各々は前記データ・ストリーム中の以前に与えられたコンテキスト・データ・セグメント関連付けられた値を持っている；

(b)コンテキスト・データ・セグメントとの間で予め定められた一時的な関係を有している現データ・セグメントにアクセスする；

(c)前記現データ・セグメントをコンテキスト・データ・セグメントに等しい値のキャッシュ識別子が割り当てられているキャッシュ記憶中に負儒まれ手いるデータ・セグメントと比較するが、前記割り当てられたキャッシュ識別子を持っているキャッシュ記憶が存在しない場合には前記キャッシュ識別子を持ったキャッシュ記憶を作る；

(d)前記現データ・セグメントがキャッシュ記憶中にあった場合には前記キャッシュ記憶のデータ・セグメントの位置識別子を送出し、前記現データ・セグメントが前記キャッシュ記憶中になかった場合には見つからなかった旨の識別子及び前記現データ・セグメントを送出する；

(e)現データセグメントが以降の現データ・セグメントに対しての新たなコンテキスト・データ・セグメントになるようにする；

(f)他のどのキャッシュ記憶にも前記新たなコンテキスト・セグメント値が割り当てられていない場合には前記新たなコンテキスト・セグメント値を新たなキャッシュ記憶に割り当て、処理すべきデータ・セグメントがそれ以上なくなるまでステップ(b)ないしステップ(f)を繰り返す。

【請求項６】以下のステップ(a)ないし(f)を設け、ラスタ・スキャン表現の２レベルのデータを複数のキャッシュ記憶を使用して圧縮し、各キャッシュ記憶は前記ラスタ・スキャン表現からの複数のデータ・セグメントを使用状況の階層に基づいてストアするデータ圧縮方法：

(a)キャッシュ・コンテキスト識別子を前記複数のキャッシュ記憶の各々に割り当てる：前記キャッシュ記憶の個数は前記データ・セグメントが何通りの値を取り得るかに等しく、各キャッシュ・コンテキスト識別子はデータ・セグメントの取り得る値と等しい値を持っている；

(b)第１のスキャン行の次のスキャン行に含まれている現データ・セグメントにアクセスする：前記現データ・セグメントの各々は前記第１のスキャン行中のコンテキスト・データ・セグメントに位置的に整列している；

(c)前記現データ・セグメントの各々を位置的に整列しているコンテキスト・データ・セグメントと同じ値が割り当てられたキャッシュ記憶中に含まれるデータ・セグメントと比較する；

(d)前記キャッシュ記憶中で見出されたデータ・セグメントの識別子位置を送出し、もし前記現データ・セグメントが前記キャッシュ記憶中になかった場合には見つからなかった旨の識別子及び前記現データ・セグメントを送出する；
(e)前記次のスキャン行に含まれる現データセグメントが次のスキャン行のデータ・セグメントのコンテキスト・データ・セグメントになるようにする；
(f)処理すべきスキャン行がそれ以上なくなるまでステップ(b)ないしステップ(f)を繰り返す。
【請求項７】以下のステップ(a)ないし(f)を設け、データ・セグメントのストリームを複数のキャッシュ記憶を使用して圧縮し、各キャッシュ記憶は前記データ・ストリームからの複数のデータ・セグメントを使用状況の階層に基づいてストアするデータ圧縮方法：
(a)キャッシュ・コンテキスト識別子を前記複数のキャッシュ記憶の各々に割り当てる：前記キャッシュ記憶の個数は前記データ・セグメントが何通りの値を取り得るかに等しく、各キャッシュ・コンテキスト識別子はデータ・セグメントの取り得る値と等しい値を持っている；
(b)コンテキスト・データ・セグメントとの間で予め定められた一時的な関係を有している現データ・セグメントにアクセスする；
(c)前記現データ・セグメントをコンテキスト・データ・セグメントに等しい値のキャッシュ識別子が割り当てられているキャッシュ記憶中に負儒まれ手いるデータ・セグメントと比較する；
(d)前記キャッシュ記憶中で見出されたデータ・セグメントの識別子位置を送出し、もし前記現データ・セグメントが前記キャッシュ記憶中になかった場合には見つからなかった旨の識別子及び前記現データ・セグメントを送出する；
(e)処理すべきデータ・セグメントがそれ以上なくなるまでステップ(b)ないしステップ(f)を繰り返す。
【請求項８】以下の(a)ないし(c)を設けたデータ圧縮システム：
(a)複数のメモリ位置に複数のデータ・セグメントをストアする第１のキャッシュ・メモリ手段：前記位置は位置識別子を有する；
(b)複数のレベルを有する第２のキャッシュ・メモリ手段：前記レベルのうちの第１のレベルは最も最近使用されたデータ・セグメントについての位置識別子を含み、前記位置識別子は前記第１のキャッシュ・メモリ手段中のどこに前記最も最近使用されたデータ・セグメントが存在し、前記レベルのうちのそれ以外のレベルは前記第１のキャッシュ・メモリ手段中のどこに前記最も最近使用されたデータ・セグメントよりも長い間アクセスされていないデータ・セグメントが存在するかを示す位置識別子を含む；
(c)受信したデータ・セグメントが前記第１のキャッシュ・メモリ中に見つかったか否かを判定し、もし見つかった場合には前記第１のキャッシュ・メモリ手段中で前記受信されたデータ・セグメントをストアしている位置を示す前記位置識別子を前記第２のキャッシュ・メモリ手段から得て受信手段へ送出する。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】本発明はデータの圧縮及び伸長(decompression)のための方法及び装置に関するものであり、より詳細には、同期動作するキャッシュ・メモリによるこのような方法及び装置に関するものである。
【０００２】
【従来技術及びその問題点】データの圧縮の目標は、データまたはイメージを表すために必要なビットの数を節減することにある。先行技術はデータ圧縮のための多くの方法で満ちている。最高のレベルのデータ圧縮を与える技術は一般的には最も複雑なデータ処理機器を必要とするものであり、またその実行速度が遅いものが多い。比較的低いレベルのデータ圧縮を与える技術は、比較的迅速に動作し、比較的単純なハードウエアを用いることが多い。一般的にいえば、データ圧縮の方法を選択することは、システムの複雑性や実行時間とデータ圧縮の所望のレベルとの間での妥協に基づくものである。
【０００３】刊行された先行技術には、データ圧縮方法についての多くの技術が含まれている。"Coding of Two-Tone Images", Hwang, IEEE Transactions on Communications, Vol. COM-25, No. 11, November, 1997, pp. 1406-1424 は、英数字データ及びイメージ・データの双方について効率的なコーディングのための多くの技術を説明する評論論文である。１次元コーディング（ラン・レングス）及び２次元コーディング（例えば、ブロック当りのピクセル・データ(per block of pixeldate)）の双方について考察されている。Hunter 他は論文 "International Digital Facsimile Coding Standars", Proceedings of the IEEE, Vol. 68, Nor. 7, July, 1980, pp. 854-867 で、ファクシミリ通信において用いられる種々のアルゴリズム（一般的には、１次元コーディング技術）を説明している。彼らはまた、後続のコーディング・ラインの条件が先行の基準ラインにおける条件に依存してコーディングされる２次元のコーディング方式についても説明している。
【０００４】Williams による論文 "An Extremely Faxt Ziv-Lempel Data Compression Algorithm", Proceedings of the IEEE Data Compression Conference, April, 1991, pp. 362-371 には、周知の Lempel -Ziv)(LZ)の圧縮アルゴリズムを高速で実現することが説明されている。この方法では、受信ノードと送信ノードの双方でデータ・ストリングのストリングの辞書を構築し、入力されたデータ・ストリングと辞書内に見出されるデータ・ストリングとの間の一致に従ってコードを送出する。

【０００５】Usubuchi 他は論文 "Adaptive Predictive Coding For Newspaper Facsimile", Proceedings of the IEEE, Vol. 68, No. 1980, pp. 807-813 で、ハーフ・トーンのイメージ・データの圧縮に適用される適応的、予測的なアルゴリズムを説明している。ハーフ・トーンの画像イメージをコーディングする更に別の予測的な方法については、Stoffel による論文 "Half-tone Pictorial Encoding", SPIE Applications of Digital Image Processing, Vol. 19, 1977, pp. 56-63 で説明されている。Stoffel のアルゴリズムは、イメージをブロックに分割し、先行のブロックから現ブロックを予測することを試みる。イメージの最終的なコーディング結果は予測誤差とブロックの値からなる。

【０００６】Langdon, Jr. 他の論文 "Compression of Black-White Images with Arithmetic Coding", IEEE Transactions on Communications, Vol. COM-29, No. 6, June, 1981, pp. 858-867 では、ピクセル毎の確率をピクセル・コンテキスト（すなわち周辺のピクセル））に基づいて推定する算術的コーディング方法が説明されている。Langdon, Jr. 他の算術的コードでは、ある種の古くからの算術的コードに固有の乗算操作を回避している。Hampel 他による論文 "Technical Features of the JBIG Standard for Progressive Bi-Level Image Compression", Signal Precessing: Image Communicatin Journal, Vol. 4, Nor, 2 (1992), pp. 103-111 に示されているように、Langdon, Jr. 他の圧縮技術は間もなく２レベル・イメージ・データのコーディングの国際的な標準になるはずである。

【０００７】Bentley 他は論文 "A Locally Adptive Data Compression Scheme", Communications of the ACM, April 8, 1986, Vol. 29, Nor. 4, pp. 320-330 及び "Technical Correspondence", Communications of the ACM, September 1987, Vol, 30, No. 9, pp. 792, 793 で、自己組織型逐次サーチ技術を用いてテキスト・データを圧縮する方法を説明している。具体的に言えば、頻繁にアクセスされるワードはサーチ・シーケンスの先頭の近傍にあり、このためにそれらのワードは圧縮動作に当たって短時間で見つかるということである。

【０００８】Bentley 他の論文で説明されているシステムは、中央処理ユニットのメモリ・システム中のキャッシュ・メモリに類似している。具体的に言えば、ＬＲＵ（least recently used）管理を使ったキャッシュ・メモリ（数のリストの形態をとるキャッシュ）が採用されている。このリストは、最も最近使用されたものからこれまで一番長い期間使用されていないものへという態様で順序付けられる。サーチ対象の値つまりワードがこのリスト中にあれば、その度に、このワードがキャッシュ・リストから取り出されてこのキャッシュ・リストの先頭に配置され、残りの値は１つずつ後ろにずらされる。キャッシュ・リスト内に入っていないワードがサーチ対象であったときは、もっとも長い間使用されていなかった値がキャッシュから除去されて、新規の値を受け入れる余地を（リストの先頭に）作成する。

【０００９】図１のテーブルには、このようなキャッシュの動作シーケンスが例示されている。テーブル２０に例示された４位置のキャッシュにおいて、最も最近使用された値はキャッシュの「先頭」にあり、最も長い間使用されていなかった値はキャッシュの「底」にあり、またその他の値は中間の位置にある。２２、２４及び２６という参照番号の付いたキャッシュには、キャッシュ２０の連続的な状態が例示されている。キャッシュの状態２２はキャッシュ２０の初期状態であり、それには４個の１０進値がストアされて示されており、１０進値１０が最も最近使用された値であるとされている。キャッシュの状態２４は、値６をサーチした後のキャッシュ２０の状態である。この値６はキャッシュ２０内に既に存在することから、値１０及び５がプッシュ・ダウンされる。値２５をサーチした後が遭遇された後、キャッシュの状態２６が生じる。この値２５はキャッシュ２０内に存在しないことから、この値はキャッシュ２０の先頭に挿入され、最も長い間使用されていなかった値８が取り除かれ、また、その他の値はプッシュ・ダウンされる。

【００１０】データの圧縮及び伸長のために採用されるコーディング・プロセス及びデコード・プロセスでは、キャッシュの状態がエンコーダ機構及びデコーダ機能の双方と確実に同期するように、圧縮されたコード・ワードとキャッシュの適応の操作がなされる。キャッシュの同期をとることにより、欠落のないデータ処理が確実に行われるようになる。図１に示されているように、キャッシュ２０の状態は、新たな値についてのサーチが行われる毎に「適応される」。ある値がエンコーダ内のキャッシュに既に存在しているときには、そのキャッシュ位置が、この値の位置を指示するコード・ワードの形式で送出される。この値がキャッシュ内に存在しないときには、ある特別なコード・ワードがこの値それ自体とともにエンコーダによって出力される。通常は値それ自体よりも少ないビット数でキャッシュ位置を送出できるために、圧縮が達成される。デコーダでは、受取ったコード・ワードを解釈して、当初のデータ・ストリームを再構成する。更に、このデコーダはエンコーダと同様のやり方でそのキャッシュを「適応させて」、エンコーダとの同期状態を維持する。

【００１１】図２及び図３には、エンコーダ機構及びデコーダ機構の双方で同期を確保するために採用される手続のフローチャートが図示されている。図２にはコーディング手続が例示されており、また、図３にはデコード手続が例示されている。図２及び図３の各ブロック中の処理は以下の通りである：

図２

３０：圧縮されていない次のデータ値を獲得

３２：キャッシュ中の値と一致するか？
３４：キャッシュ中の位置を示すコード・ワードを出力
３６：キャッシュを適応させる
３８：「発見せず」を示すコード・ワードを出力
４０：データ値を出力
４２：この新たな値をキャッシュに挿入し、キャッシュを適応させる
図３
４４：圧縮されている次のコード・ワードを獲得
４６：キャッシュ中の位置を示すコード・ワードか？
４８：キャッシュ値を出力
５０：キャッシュを適応させる
５２：データ値を獲得して出力
５４：この新たなデータ値をキャッシュに挿入し、キャッシュを適応させる
ここで図２を参照すると、圧縮されていないデータ値（例えば、１バイト）をアクセスし（ブロック３０）、この圧縮されていないデータ値が圧縮キャッシュ内の値に一致しているものがあるかどうかの判定がなされる（判定ブロック３２）。判定結果がｙｅｓであるときには、このキャッシュ中で一致した値の位置を指示する位置コードが出力される（ブロック３４）。必要である場合には、一致したキャッシュ中の値を当該キャッシュの先頭まで移動しこれに応じて残りの値を再配置するように移動させることによってキャッシュの適応がなされる（ブロック３６）。そして、この手続は、次の圧縮されていないデータ値を獲得するところまで戻って再び繰り返される。

【００１２】受信したある値に一致するものがキャッシュ中で発見できなかった場合には、「発見せず」、つまりミスヒット、を表すコード・ワードが出力され（ブロック３８）、また実際のデータ値が出力される（ブロック４０）。そして、受信したデータ値がキャッシュの先頭に挿入され、また、残りの値をプッシュ・ダウンし、最も長い間使用されていなかった値を排除することによって、キャッシュの残りのものについての「適応」がなされる。

【００１３】伸長に当たっては（図３）、圧縮されたコード・ワードをアクセスし（ブロック４４）、このコード・ワードに位置コード・ワードが含まれているかどうか判定する（判定ブロック４６）。判定結果がｙｅｓであるときには、デコーダのキャッシュ中の当該位置の値を出力し（ブロック４８）、また出力されたキャッシュ値をキャッシュの先頭まで移動させることにより、このキャッシュの適応がなされる（ブロック５０）。

【００１４】上述した場合とは逆に、受信したデータが位置コード・ワードではなかったときには、この受信したデータ値をアクセスして、出力する（ブロック５２）。また、このデータ値はまたキャッシュの先頭に挿入され、更に、このキャッシュの残りの部分についての適応が成される。

【００１５】上述したようなキャッシュ・ベースの圧縮手続は効果的なものではあるが、ＬＲＵ方式のキャッシュ管理は演算が複雑であることが多い。キャッシュを「適応」させる度に多くの値を再配置することが必要とされる。このような再配置は、一連の圧縮の間に何回も生起するために、相当な処理時間を占める可能性があり、その圧縮手続の効率を著しく低下させる。

【００１６】
【目的】従って、本発明の目的は、改良されたキャッシュ管理技術を使用することにより、キャッシュ・ベースの圧縮手続をより効率的にすることにある。

【００１７】本発明の更に別の目的は、キャッシュ・ベースの予測技術を用いた、改良された圧縮／伸長手続を提供することにある。

【００１８】本発明のなおも別の目的は、イメージの処理に特に適応しており、圧縮されるべき値の近傍におけるデータの内容を用いるようにする、キャッシュ・ベースの圧縮／伸長手続を提供することにある。

【００１９】
【発明の概要】本発明の一実施例によれば、２レベルのデータを圧縮するためのシステムには、使用が割り当てられた複数のレベルを有する第１のキャッシュ・メモリが含まれている。その第１のレベルは最も使用頻度が高いデータ・セグメントに対して割り当てられ、その第２のレベルは複数のより使用頻度が低いデータ・セグメントに対して割り当てられている。プロセッサは、受信したデータがデータ・セグメントがキャッシュ・メモリ内で発見されたかどうかを判定し、発見されなされなかったときには、この受信したデータ・セグメントを、キャッシュ・メモリの第１のレベルに既にストアされている以前のデータセグメントに代えてそこに割り当てる。この以前のデータ・セグメントは、第２のデータセグメントに以前にストアされているところの、より長い間使用されていなかったデータ・セグメントに代えて、その位置に割り当てられる。より長い間使用されていなかったデータ・セグメントであって置換されるものは、疑似ランダム法によって選択される。そして、発見されなかったという通知は、受信したデータ・セグメントの識別情報とともに、受信ステーションに送出される。この受信ステーションは送信側と同じキャッシュ構造を含んでおり、受信されたコード・ワード及びデータ・セグメントに応答してそのキャッシュを更新する。本発明の更に別の変形によれば、データ・セグメントのまわりのコンテキストに従って、複数のキャッシュ・メモリに対してデータ・セグメントが入力される。なお、これら複数のキャッシュ・メモリは、コンテキストから導出されたアドレスに従って割り当てられる。

【００２０】
【実施例】これ以降の本発明の説明は、プリンタとの通

信をするホスト・コンピュータに関連してなされるものであり、そのコーディングは、送出されるデータのバイト（８ビット）に対して行われる。しかしながら、後述する手続及びシステムは、データを送り出し、また送出時のバンド幅の経済性やメモリの節約のために圧縮／伸長の手続を採用する多様なシステムに適用されるということを理解しなければならない。

【００２１】図４において、ホスト・コンピュータ６０には、ホスト中央処理ユニット（ＣＰＵ）６２、リード・オンリ・メモリ（ＲＯＭ）６４及びランダム・アクセス・メモリ（ＲＡＭ）６６が含まれている。これらのモジュールの各々はバス６８で相互接続されており、その入力／出力の通信はＩ／Ｏモジュール７０によって処理される。ＲＯＭ６４には、Ｉ／Ｏモジュール７０が送出ライン上にデータを送出する前にホストＣＰＵ６２の制御の下にデータの圧縮操作を行うコード化手続が含まれている。ＲＡＭ６６は、ＬＲＵ方式で構造化され、データの圧縮操作を達成できるようにする複数のキャッシュ・メモリを含む。

【００２２】Ｉ／Ｏモジュール７０から送出されたデータは、Ｉ／Ｏモジュール７３を介してプリンタ７２で受信される。プリンタ７２には、ＣＰＵ７４、プリント・エンジン７６、ＲＡＭ７８及びＲＯＭ８０が含まれている。ＲＡＭ７８は、Ｉ／Ｏモジュール７３を介して受信された入力データのための一時記憶部として作用するものであり、また、ホストＣＰＵ６０中のＲＡＭ６６に含まれているキャッシュ・メモリと構造的に同一のキャッシュ・メモリも含んでいる。ＲＯＭ８０は、Ｉ／Ｏモジュール７３を介して受信した圧縮済みデータを伸長し、使用するためにＲＡＭ７８中にストアするようにするデコード手続を含んでいる。

【００２３】先に示したように、ＲＡＭ６６及び７８に含まれているキャッシュ・メモリはＬＲＵキャッシュとして構成されている。このキャッシュでは、キャッシュの先頭のデータは最も最近使用された（ＭＲＵ）ものであり、このキャッシュの底にあるデータは最も長い間使用されなかったものである。キャッシュが更新される（「適応させられる」）度に多くの値の再配置が必要とされることから、このようなキャッシュの管理は演算上複雑なものである。

【００２４】図５に、演算上の複雑性を減少させるような、修正された形式のＬＲＵキャッシュを示す。キャッシュ９０は階層化されたＬＲＵキャッシュとして構成されており、ここに、最上位層９１は最も最近使用されたデータ・バイトであり、層９２～９４は３個の最も長い間使用されなかった（ＬＲＵ）データ・バイトを含む「単一の」層を有している。ここで、単に説明を簡単にするだけの目的で、キャッシュ９０が４個のエントリと２つの層を有するものとして示されていることを理解されたい。任意の数の層を設けまた層毎に任意の個数のエントリを有することができる。

【００２５】キャッシュ９０の特定の層からどの値を選んで取り除くかということは、コード化機構とデコード機構との間の同期状態を維持しながら、疑似ランダム的に行うことができる。例えば、ラウンド・ロビン方式カウント法を用いることができる。このやり方では、最も長い間使用されなかった層から取り除かれるべきバイトは、エントリ９２～９４の逐次的なカウントに基づいて選択される。この態様において、キャッシュが大きくなると、処理上のオーバヘッドをより少なくして管理でき、またそれでもＬＲＵキャッシュと殆ど同様に効率的である。

【００２６】キャッシュの状態表示Ａ、Ｂ及びＣでは、３つの異なる時点（すなわちＡ、Ｂ及びＣ）におけるキャッシュ９０の状態が示されている。時点Ａにおいては、キャッシュの状態は図示した通りである。時点Ｂにおいて値２のバイトがサーチされ、最も最近使用された層９１に置かれる。値２のバイトは既にキャッシュの第２のレベルにあることから、第１のレベルの値７８とその位置を交換し、これによって（キャッシュの状態９６によって示されるように）値７８を位置９４に置く。時点Ｃにおいて、値８９のバイトをサーチするものとする。この値はキャッシュ中にはないことから、最も最近使用されたものを入れておく層９１に挿入され、また最も長い間使用されなかったものが入る層（この場合では、位置９３）から疑似ランダムに導出された位置には、値８９によって置換された値（すなわち値２）が上書きされる。

【００２７】最も長い間使用されなかった層での交換が次回に必要とされた際には、置換されるのは位置９４であり、これに次いで位置９２、位置９３等々となる。従って、キャッシュ９０を階層化することにより、適応操作の間のキャッシュ値の移動を少なくすることができる。構成的にいえば、最も長い間使用されなかったものを入れておく層中で交換されるべき位置は、ラウンド・ロビン方式のカウントが増大するにつれて更新されるポインタの位置に基づいて決定することができる。

【００２８】新規に受信したバイトを入れるべき多くの独立したキャッシュの中から選択するためのコンテキストを採用することにより、パフォーマンスを更に向上することができる。プリンタには通常、各ピクセルが２値のビット値で表されるところの、ピクセルのラスタ・イメージを有するビット・マップ・メモリを備えている。各ラスタ・スキャン行は８ビット・バイトのシーケンスに分割される。前以ってデコードされているような位置に置かれている周辺のピクセルを、最も最近使用されたバイトをストアするためにいずれのキャッシュを用いるべきかを判定するコンテキストとして用いる。周辺のピクセルのビット値を組み合わせて１つの数を形成し、どのキャッシュを使用すべきかを指定するコンテキスト値

（つまりインデクス）が導出できるようにする。この操作を行うことで、各キャッシュが個々のコンテキストに対してより良好に適応できるようにすることにより、データ圧縮の改善がなされる。

【００２９】図６中の図において、ラスタ走査ラインの番号の付いた８個のピクセル（ピクセル１００）は、コード化される８個のハッチングの付いたのピクセル（ピクセル１０２）のためのコンテキスト値を形成する。これ以降では、ピクセル１００をコンテキスト・バイトと呼び、またピクセル１０２は現在バイトと呼ぶ。コンテキスト依存キャッシュ構成の思想は、イメージは垂直方向に相関関係を持っている（例えば、表組みにおける垂直の線）ことが多いという事実に基づいている。また、キャッシュのアドレスとしてコンテキスト・バイトを採用することにより、現在バイトでサーチした際、アドレスされたキャッシュ内の最も最近使用されたものが入るエントリ中に現在バイトと同一のデータ・バイトが入っている可能性が高くなり、これにより、キャッシュのデータをそれ以上移動させることなく、高速にコード化することができるようになる。「垂直の」コンテキスト・バイトは後続する説明の基本を形成するものであるが、別の物理的な関係も本発明の範囲内に入っているということを理解しなければならない。

【００３０】コンテキスト依存キャッシュの構成を実現したものでは、圧縮／伸長手続の開始時点で全てのキャッシュを生成し初期化してよい。例えば、８ビット・バイトが採用されるときには、最初に２５６個のキャッシュを生成し、各キャッシュには複数のエントリを持てるようにしておくことができる。キャッシュが複雑化することによる問題よりもメモリの方が重要であるときには、全てのキャッシュを最初に生成しておく必要ではなく、必要とされるときにのみキャッシュを生成させることができる。

【００３１】図７は、データ・セグメントに対する圧縮コード化手続を説明するフローチャートである。このフローチャートの各ブロックの動作は以下の通りである：

１０４：圧縮されていない次のデータ値を獲得
１０６：コンテキスト値に基づいてキャッシュを選択
１０８：キャッシュ値が一致したか？
１１０：その位置を表すコード・ワードを出力
１１２：キャッシュを適応させる
１１４：「発見せず」を示すコード・ワードを出力
１１６：データ値を出力
１１８：この新たなデータをキャッシュに挿入しキャッシュを適応させる

図７に示されているように、送出されるべき新たなデータ・バイトが与えられると（判定ブロック１０４）、そのピクセル・イメージの直前のスキャン行に垂直に整列したコンテキスト・バイトのアドレスを有する特定のキャッシュがアドレスされる（ブロック１０６）。（先に示したように、このアドレスされたキャッシュ中の最も最近使用されたエントリは、与えられた圧縮されていないバイト値と同じである可能性が高い。）次に、このバイト値をこのアドレスされたキャッシュ中の値と比較する（判定ブロック１０８）。ここで一致が取れた場合には、アドレスされたキャッシュ中で一致した値の位置を指示するコード・ワードを出力する（ブロック１１０）。また、必要ならその一致した値をアドレスされたキャッシュ中の最も最近使用されたものが入るエントリに持ち上げ（ブロック１１２）、この手続の先頭に戻る。

【００３２】アドレスされたキャッシュにおいて一致が取れなかった場合には（判定ブロック１０８）、「発見せず」、つまりミスヒット、を表すコード・ワードを出力し（ブロック１１４）、次いで実際のバイト値を送出する（ブロック１１６）。このキャッシュ中のどの値とも一致しなかったバイト値はアドレスされたキャッシュに送出され、このキャッシュを適応させて（ブロック１１８）、またこれによって置き換えられたバイトをキャッシュのもっと下位のレベルに移動させ、更にこのもっと下位のレベルに既にストアされていたバイトを置換する等の作業がなされる。次に、この手続はその先頭に戻る。

【００３３】図８は、図７のフローチャートに従ってコード化されたデータをデコードするための伸長手続を説明するフローチャートである。このフローチャート中の各ブロックの動作は以下の通りである：

１２０：圧縮されている次のコード・ワードを獲得
１２２：コンテキスト値に基づいてキャッシュを選択
１２４：キャッシュ中の位置を示すコード・ワードか？
１２６：キャッシュ値を出力
１２８：キャッシュを適応させる
１３０：データ値を獲得して出力
１３２：この新たなデータ値をキャッシュに挿入し、キャッシュを適応させる

図８には、図７に示されている手続で圧縮されたデータに応答する伸長の手続が示されている。受信された圧縮済みコード・ワードがアクセスされ（ブロック１２０）、直上のスキャン行中の既にデコードされたコンテキスト・バイトのコンテキスト値に基づいてキャッシュがアドレスされる（ブロック１２２）。圧縮されたコード・ワードがキャッシュ中の位置を示すコード・ワードである場合には（判定ブロック１２４）、アドレスされたキャッシュ中の指示された位置に現れている値を出力する（ブロック１２６）。また、必要なら最も最近使用されたバイトをキャッシュの先頭に置くことができるようにそのキャッシュ内のエントリを再配置することにより、キャッシュを適応させる（ブロック１２８）。

【００３４】受信したデータがキャッシュ中の位置を示すコード・ワードではないと判定された場合には（判定

ブロック１２４）、この受信したバイト値がメモリから取り出して出力する（ブロック１３０）。また、当該新たなバイト値をアドレスされたキャッシュの先頭に挿入し、このキャッシュを適応させる。更に、受信したデータがもうなくなるまでこの手続を繰り返す。

【００３５】これを要約すると、現在行バイトは、そのアドレスが現在バイトの直上のコンテキスト・バイトの値であるキャッシュに常に振り向けられる。次のスキャン行がアクセスされるときには、現在バイトがコンテキスト・バイトになる。このコンテキスト・バイトの値が既にキャッシュのアドレスであるときには、新たにキャッシュを生成する必要はない。コンテキスト・バイトの値が新たなものであるときには、コンテキスト・バイトをそのアドレスとして、新たなキャッシュが設定される。初期化時に全てのキャッシュが設定されている場合には、出現し得るコンテキスト・バイトの値毎に１つのキャッシュが存在するので、新たにキャッシュが要求されることはない。

【００３６】上述の議論では、コンテキスト値の概念が導入された。この議論でコンテキスト値を構成するものは、現在ラインの直上のラスタ・スキャン・ラインからのデータ・セグメントの値である。このコンテキスト値はキャッシュのアドレスとして用いられた。このコンテキスト値は先にデコードされたいくつかのデータ・セグメントの組み合せから形成することもできる。そのようにした場合の利点は、コンテキストを大きくすると、現在のデータ・セグメントをコード化するための最も性能の良いキャッシュを選択するより良い手段を提供できることがあるという点である。このコンセプトの１つの具体的な実現形態では、現在データ・セグメントのすぐ上のデータ・セグメントだけではなく、図１４に示されているように、現在のデータ・セグメントのすぐ上のものの左側にあるデータ・セグメントを使用する。これら２個のデータ・セグメントを組み合わせてコンテキスト値にする１つのやり方は、現在データ・セグメントのすぐ上のものの左側にあるデータ・セグメントから取り出した２つのビットを、現在データ・セグメントのすぐ上のデータ・セグメントの左側へ付加することである。図１４で“A”なるラベルが付されたバイトは現在データ・セグメントである。“B”及び“C”なるラベルが付されたバイトはコンテキスト値を形成するために用いられるデータ・セグメントである。コンテキスト値は、データ・セグメントＢのビット１及び２をデータ・セグメントＣのビット１ないし８に連結することによって発生でき、これによって１０ビット・データのコンテキスト値が得られる。

【００３７】あるシステムでは、より複雑なコード化手続を設けるという犠牲を払ってデコード／伸長手続の複雑性を減少させることが望まれる。間接的キャッシュ管理を用いることにより、デコード手続の一部をコード化手続に移管することができる。ＬＲＵキャッシュの場合には、新たに受信されたデータ・バイトがキャッシュ内に既に記憶されていることが発見される限り、キャッシュ内の値が変化することはなく、再配置が行われるだけであるという事実により、この手続の一部の移管が可能にされる。ＬＲＵキャッシュ内の値のセットが変化する唯一の時点は、新たなデータ・バイトとキャッシュ内に既にストアされているバイトとの間に一致が見出されない場合だけである。間接的キャッシュ管理においては、エンコーダを用いて複雑さを増すという犠牲を払って、ＬＲＵキャッシュを維持するために必要な値の再配置をデコーダがしなくても良いようにしている。

【００３８】図９を参照すると、２つの番号リスト、すなわちキャッシュ・メモリ１２０内のポインタ・リスト及びキャッシュ・メモリ１２２内のバイト値リスト、を維持・管理するためのコード化手続ができるようにする一対のキャッシュ・メモリ１２０及び１２２が示されている。ポインタ・リストには、値リスト内のエントリへのポインタつまりインデクスが含まれている。新たに受信されたバイトとキャッシュ・メモリ１２２内のバイト値リストに既に記憶されているバイトとの間で一致が見出されたときには、キャッシュ・メモリ１２０内のポインタ・リスト中のポインタはＬＲＵ方式で再配置される。新たな値がキャッシュ・メモリに入力されておらず、また古い値がキャッシュ・メモリから取り除かれていないため、値リスト内の値は再配置を必要としない。（新たなバイトとキャッシュ・メモリ１２２内に既にストアされているバイトとの間で）一致するものがあることがわかると、エンコーダからそれに対応するポインタ値が発せられる（すなわち、エンコーダは、受信されたバイトに対応するキャッシュ・メモリ１２２内の値エントリを指示する）。

【００３９】キャッシュ・メモリ１２２内で一致するものが見出されないときには、値リスト及びポインタ・リストは修正されねばならない。キャッシュ・メモリ１２２内の値リスト内で最も長い間使用されなかった値が取り除かれる。この操作は、キャッシュ・メモリ１２２内の値リスト位置のうちの、キャッシュ・メモリ１２０内にある最も長い間使用されなかったものを示すポインタ（ＬＲＵポインタ）が指している位置に現在バイト値を置くことによって達成される。次に、キャッシュ・メモリ１２０内のポインタは、キャッシュ・メモリ１２２内の新たな値の順序を反映するように更新される。この新たに入力された現在値は、キャッシュ・メモリ１２０内にある最も最近使用されたものを示すポインタ（ＭＲＵポインタ）によって指し示される。この場合、エンコーダは一致がなかったことを知らせる特別なコード・ワードを発し、次に実際のバイト値及び（デコーダ中の）値リスト中のこの実際のバイト値が挿入されるべき位置へのポインタをその後に供給する。

【００４０】図９において示されているように、キャッシュ・メモリ１２０におけるポインタ・リストのエントリは、最も最近使用されたもの（ＭＲＵ）から最も長い間使用されなかったもの（ＬＲＵ）への順序で配置される。かくして、キャッシュ・メモリ１２２内の値リスト中のバイト値の位置に係わらず、ポインタ・リスト内のポインタの位置から、最も最近使用されたバイト値が値リスト内のどこにあるかがわかり、またこのバイト値以外のストアされているバイト値についてのＭＲＵ以降の全ての使用レベルがわかる。

【００４１】デコーダでは、ポインタ・リストではなく、値リストだけが維持・管理される。コード・ワード（例えばポインタ値）がデコードされると、値リスト中の指し示された位置から読み出した値が出力される。「発見せず」というコード・ワードを受信した場合には、ポインタ及びその値が入力ストリームから導出されて、この値がデコーダ内の値リスト中の指し示された位置に置かれる。

【００４２】図１０を参照して、間接的キャッシュ管理のコード化（圧縮）のための手続について説明する。図１０の各ブロックの動作は以下の通りである：

１３０：圧縮されていない次のデータ値を獲得

１３２：キャッシュ値が一致したか？

１３４：その位置を表す値リスト・ポインタ・コード・ワードを出力

１３６：キャッシュを適応させる

１３８：「発見せず」を示すコード・ワードを出力

１４０：データ値を出力

１４２：値リストのＬＲＵ位置を示すインデクス・コード・ワードを出力

１４４：この新たなデータを値リストのＬＲＵ値位置に挿入しキャッシュを適応させる

新たなデータ値が与えられると（ブロック１３０）と、この値が値リストを含むキャッシュ・メモリ１２２内に既に記憶されている値と一致するかどうかの判定がなされる（ブロック１３２）。判定結果がｙｅｓであるときには、値リスト中で一致した値がどこにあるかを示すポインタ値をポインタ・リストから出力する（ブロック１３４）。次に、値リスト内の最も最近使用されたバイト値を適正に指示するように、ポインタ・リスト内のポインタを適応させる。

【００４３】キャッシュ値の一致が発見されなかったときには（判定ブロック１３２）、「発見せず」のコード・ワードを出力し（ブロック１３８）、それとともに、値スト中で一致するものが発見されなかったデータ値も出力する（ブロック１４０）。これに加えて、値リスト中で最も長い間使用されていなかった値を指示するポインタ値も出力する。かくして、新たなデータ値が値リスト内の最も長い間使用されていなかった値の位置に挿入され、その中に以前からストアされていた値は取り除かれる。次に、次に続く圧縮されていないデータ値を取得するため、この手続はその最初に戻る。

【００４４】図１１にデコード（伸長）手続を示す。図１１の各ブロックの動作は以下の通りである：

１５０：圧縮されている次のコード・ワードを獲得

１５２：ポインタ・コード・ワードか？

１５４：ポインタ位置の値を出力

１５６：データ値を獲得して出力

１５８：置換え値リスト・インデクス位置を獲得

１６０：この新たなデータ値を指定された値リスト・ポインタ位置に挿入

この手続では、最初に、圧縮されている次のコード・ワードをアクセスする（ブロック１５０）。このコード・ワードがポインタであることがわかった場合には（判定ブロック１５２）、対応する値リスト中のそのポインタが「指し示している」値を出力する（ブロック１５４）。コード・ワードがポインタ値ではないことがわかった場合には（判定ブロック１５２）、受信されたデータ値をアクセスする（ブロック１５６）。この際、置換えポインタ値も獲得する（ブロック１５８）。次に、新たなデータ値を値リスト中のこのポインタで指し示されている位置に挿入する。既にそこにある値は廃棄される（ブロック１６０）。処理すべき更に別の圧縮されたコード・ワードがなくなるまで、この手続を繰り返す。ここで、間接的キャッシュ管理と階層化キャッシュ管理を組み合わせて双方の技術に固有の長所を得ることができるということを理解すべきである。

【００４５】上述したシステムは、８ビット・バイトでデータをコーディングするものとして示されている。ある種のディザーをかけたイメージ・データのように、別の周期性、すなわち６ビット、１０ビット等、のあるデータであるときには、このようなデータをイメージ・データの周期性に一致したビット長の単位に分解してもよい。

【００４６】上述したアルゴリズムの一つの特徴は、値がキャッシュ中に入っているかどうかを判定するためには、この値をこのキャッシュ内にストアされている全ての値と比較しなければならないことである。この比較手続は、特定の値がキャッシュ内に存在するかどうかについてのフラグを含むルック・アップ・テーブルを維持・管理することによって回避することができる（図１２のルック・アップ・テーブル１７０を参照）。ルック・アップ・テーブルは可能な全ての値を含まなければならない。この手続は更に、この値がキャッシュ内に存在するときには、その位置がキャッシュ内で判定されねばならないということも必要とされる。

【００４７】上述したように、圧縮動作と伸長動作の間に確実に同期がとられるように、エンコーダとデコーダの双方におけるキャッシュは同じ初期状態でなければならない。エンコーダがある程度複雑化するという犠牲を

払って、デコーダのこの初期化を行わないようにできる。エンコーダは、（図１３のキャッシュ・メモリ１８０で示すように）全てのキャッシュ・エントリフラグを関連付けることができるさせる。このフラグは対応するキャッシュ・エントリが定義済みでないあるいは定義済みの値を持っているか否かを示す。定義済みの値とは、キャッシュ・エントリが処理されているイメージからの実際のデータ・バイト値を表している値である。エンコーダは、ある値を定義済みでないキャッシュ値を有するエントリとの比較が絶対に起こらないようにすることができる。デコーダが初期化されていない値を使用することはないので、デコーダがこれらのフラグを維持・管理する必要はない。それというのも、エンコーダが初期化されていない値を使用するようにデコーダに対して指示することがないからである。

【００４８】次の例は、２階層化されたキャッシュを使用した本発明の一実施例の動作を示す。その第１の層には一つの割り当てられた値を持っており、第２の層は３つの値を持っている。コード・ワード（ビット・シーケンス）は次のように割り当てられる。

１　　＝　第１の層の予測が正しい
０００　＝　第２の層の１番目の値の予測が正しい
００１　＝　第２の層の２番目の値の予測が正しい
０１０　＝　第２の層の３番目の値の予測が正しい
０１１　＝　訂正値が必要

キャッシュは次の条件で開始される。

第１層	第２層の１番目	第２層の２番目	第２層の３番目
0	２５５	２５５	２５５

【００４９】キャッシュの第２レベルにおけるランダムな置換は、簡単なラウンド・ロビン方式で実行される。変数"reple_index"が定義され、その初期値は１に設定される。置換が要求されたとき、この変数により、第２レベルのキャッシュ中のいずれのエントリが置換されるかが決定される。置換が実行された後、変数"reple_index"がインクリメントされる。変数"reple_index"の値が４に到達すると、その値は１にリセットされる。

【００５１】コーディングされるべきデータ・ストリームは、バイトの系列で構成されている。

【００５２】次のテーブルは、コード化シーケンスをトレースしている。キャッシュの状態は、現在の入力値が処理された後のものを示している。

【００５３】

【表１】

入力値	キャッシュ状態 第１層	第２層 No. 1	第２層 No. 2	第２層 No. 3	出力の状態及び動作
100	100	0	0	0	"011" と "01100100" が出力される キャッシュが適応させられる $repl\_num = repl\_num + 1$ (結果は 2)
17	17	0	100	0	"011" と "00010001" が出力される キャッシュが適応させられる $repl\_num = repl\_num + 1$ (結果は 3)
100	100	0	17	0	"001" が出力される キャッシュが適応させられる
100	100	0	17	0	"1" が出力される
9	9	0	17	100	"011" と "000010001" が出力される キャッシュが適応させられる $repl\_num = repl\_num + 1$ (結果は 4) $repl\_num = 1$

【００５４】ここで、前述の説明は本発明を例示しているだけであるということを理解しなければならない。当業者であれば、本発明から逸脱することなく、種々の代替及び修正を案出することができるであろう。従って、本発明は特許請求の範囲の記載の範囲内で、全てのこのような代替、修正及び変形を包含するように意図されている。

【００５５】

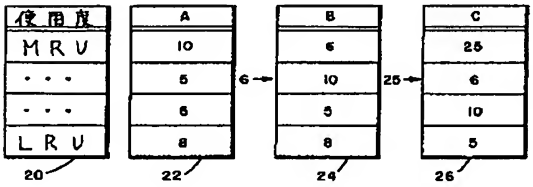
【効果】以上詳細に説明したように、本発明によれば簡単な手順でキャッシュ・ベースのデータ圧縮／伸長を行うことができる。

【図面の簡単な説明】

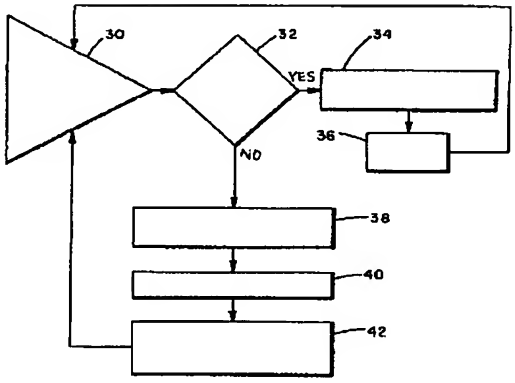
【図１】データ圧縮システムにおけるＬＲＵキャッシュについての先行技術を説明する図。

【図２】ＬＲＵキャッシュ・メモリを使用することによってデータ・セグメントをコーディングするための、先行技術によるプロセスを説明するフローチャート。
【図３】ＬＲＵキャッシュ・メモリを使用することによってデータ・セグメントをデコードするための、先行技術によるプロセスを説明するフローチャート。
【図４】本発明を実施することのできるシステムのブロック図。
【図５】階層化されたＬＲＵキャッシュを用いてデータをコード化するためのプロセスを例示する、キャッシュの状態を概略的に表示する図。
【図６】複数のＬＲＵキャッシュ・メモリの中の特定のもののアドレスとして採用されるコンテキスト・データ・セグメントを説明する図。
【図７】データ・セグメントに対する圧縮コード化手続を説明するフローチャート。
【図８】図７のフローチャートに従ってコード化されたデータをデコードするための伸長手続を説明する図。
【図９】間接的ＬＲＵキャッシュ管理を行うためにするために用いられる２個のキャッシュ・メモリを説明する図。
【図１０】図９に示されているキャッシュ・メモリに従ってキャッシュの間接管理を行うための圧縮手続を説明するフローチャート。
【図１１】図１０のフローチャートに従ってコード化されたデータ・セグメントをデコードするための伸長手続を説明するフローチャート。
【図１２】バイト・ボリュームがキャッシュ内にストアされているかどうかを指示するルック・アップ・テーブルを説明する図。
【図１３】記憶位置に対して定義された状態か定義されない状態かを指示する、各記憶位置に関連付けられたフラグを有するエンコーダ・キャッシュ・構成を説明する図。
【図１４】コンテキスト・データ・セグメントの図６とは別の配置を示す図。
【符号の説明】
６２：ホストＣＰＵ
６４：ＲＯＭ
６６：ＲＡＭ
６８：バス
７０、７３：Ｉ／Ｏモジュール
７２：プリンタ７２
７４：ＣＰＵ
７６：プリント・エンジン
７８：ＲＡＭ
８０：ＲＯＭ
９０：キャッシュ
１２０、１２２：キャッシュ・メモリ
１７０：ルック・アップ・テーブル
１８０：キャッシュ・メモリ

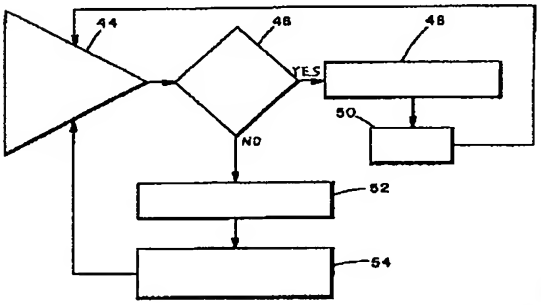
【図１】



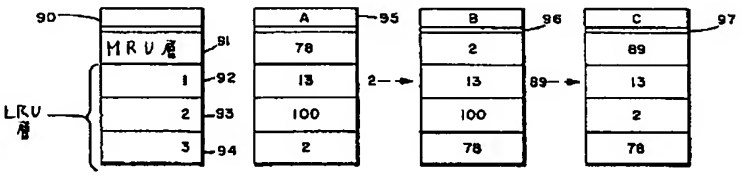
【図２】



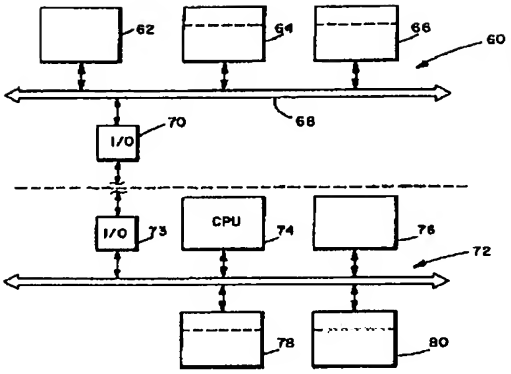
【図３】



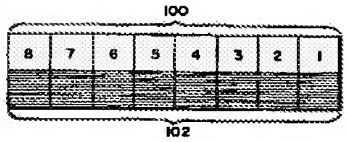
【図５】



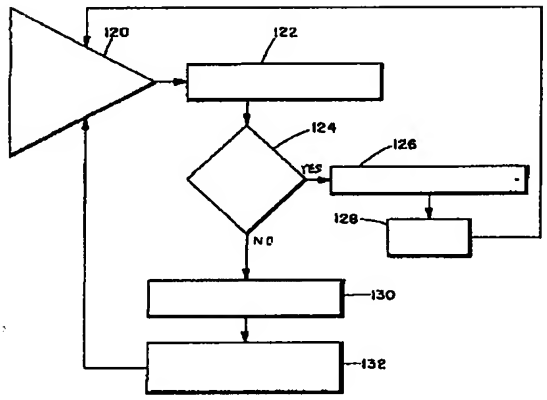
【図４】



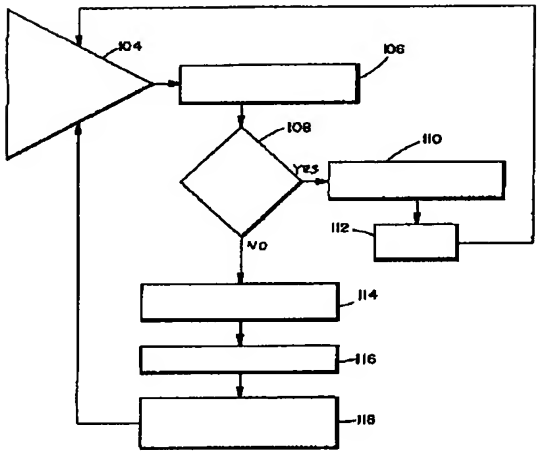
【図６】



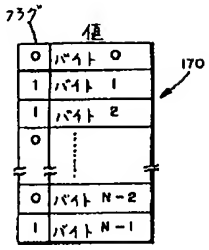
【図８】



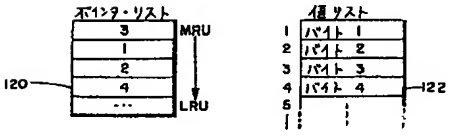
【図７】



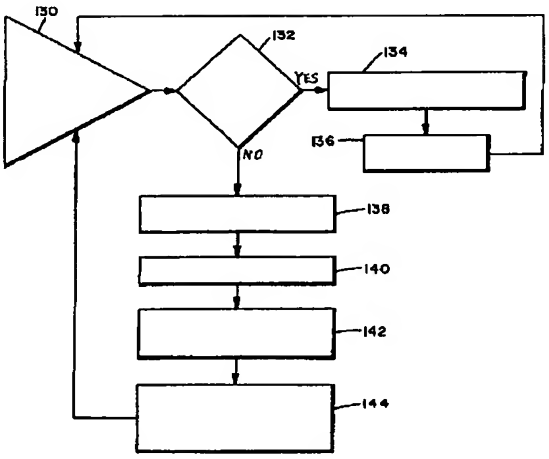
【図１２】



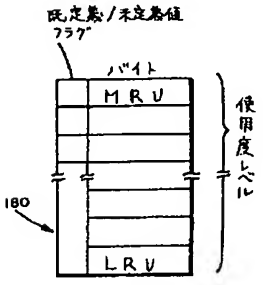
【図９】



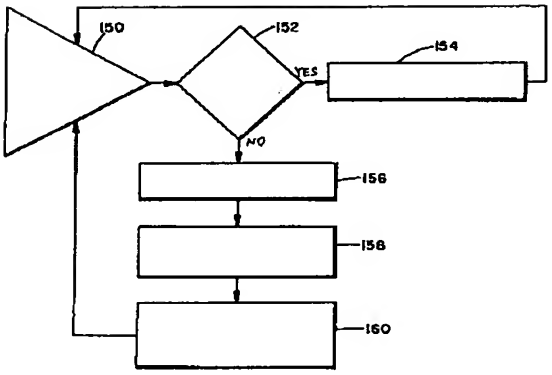
【図１０】



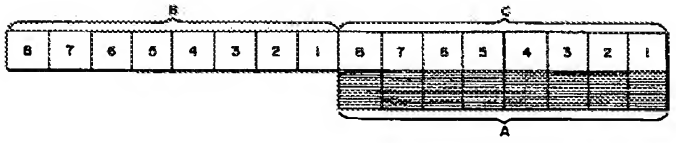
【図１３】



【図１１】



【図１４】



PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-242924
(43)Date of publication of application : 02.09.1994

(51)Int.Cl. G06F 5/00
G06F 12/12
H03M 7/30

(21)Application number : 05-283976
(22)Date of filing : 18.10.1993

(71)Applicant : HEWLETT PACKARD CO <HP>
(72)Inventor : ROSENBERG CHARLES J
BERGE THOMAS G

(30)Priority
Priority number : 92 963201 Priority date : 19.10.1992 Priority country : US

(54) SYSTEM FOR DATA COMPRESSION EXPANSION AND METHOD FOR THE SAME

(57)Abstract:
PURPOSE: To improve a problem that the load of calculating amounts for managing a dictionary is large due to complete LRU management at the time of deciding a pattern to be discarded when the number of different appearing patterns is beyond the capacity of the dictionary.
CONSTITUTION: A dictionary for appearing patterns is prepared, and when an applied patter is coincident with the pattern in the dictionary, an identifier for indicating the position of the pattern in the dictionary is outputted instead of the pattern is outputted so that data compression can be attained. The dictionary is divided into a first cache part 91 to which the most lately used pattern is inputted, and other second cache parts 92-94. When a new pattern is applied, it is inputted to the first part, and the pattern previously inputted to the first part is overwritten on a part selected at random in the second part.